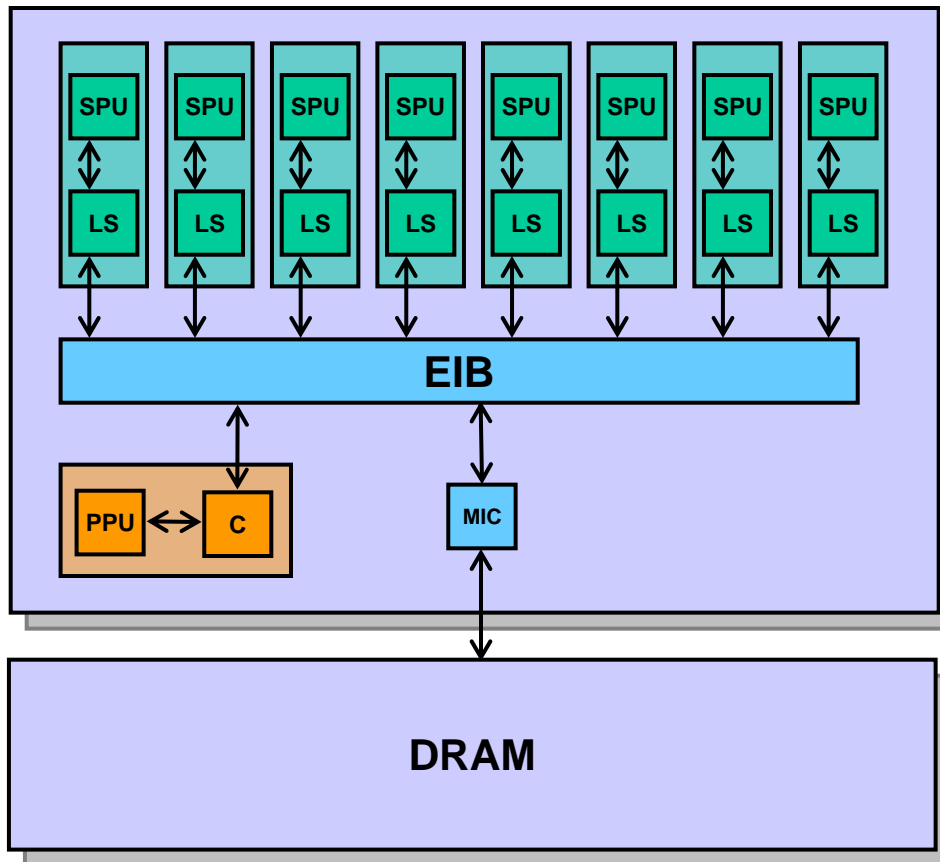




Manycore architecture of the future will be determined by compiler capabilities

**Richard A. Lethin
Reservoir Labs, Inc.**

The Challenge of Mapping to Cell



High-Level Mapping

- Partitioning between PPU/SPU
- Parallelization
- Scheduling
- Memory distribution & management
- Communication generation
- Synchronization
- Locality of reference

Low-Level Mapping

- Instruction selection
- Instruction scheduling
- Register allocation
- SIMDization

*"RISC vs. CISC" reprised
At the "High Level"*

A Compilation Flow

```
for (int i = 0; i <= 255; i++) {
  nrm[i] = 0;
  for (int j = i; j <= 255; j++) {
    nrm[i] = hypot(nrm[i], QR[j][i]);
  }
  if (nrm[i] != 0) {
    nrm[i] = sn(QR[i][i], nrm[i]);
    for (int j = i; j <= 255; j++) {
      QR[j][i] = QR[j][i] / nrm[i];
    }
    QR[i][i] = 1 + QR[i][i];
    for (int j = 1 + i; j <= 255; j++) {
      s[i][j] = 0;
      for (int k = i; k <= 255; k++) {
        s[i][j] = s[i][j] + QR[k][i] * QR[k][j];
      }
      s[i][j] = - s[i][j] / QR[i][i];
      for (int k = i; k <= 255; k++) {
        QR[k][j] = QR[k][j] + QR[k][i] *
          s[i][j];
      }
    }
  }
}
Rdiag[i] = - nrm[i];
}
```

ANSI C

**High-Level
Compiler**

API

**Low-Level
Compiler**

Cell

Partitioning between PPU and SPU

Parallelization across streaming elements

Memory layout and placement optimization

“Tiling” choosing correct size and organization of tasks

Improvements to locality of reference – producer consumer

Management of communication – intra and inter chip

Management of distributed memory

Double buffering and high-level software pipelining

Instruction selection

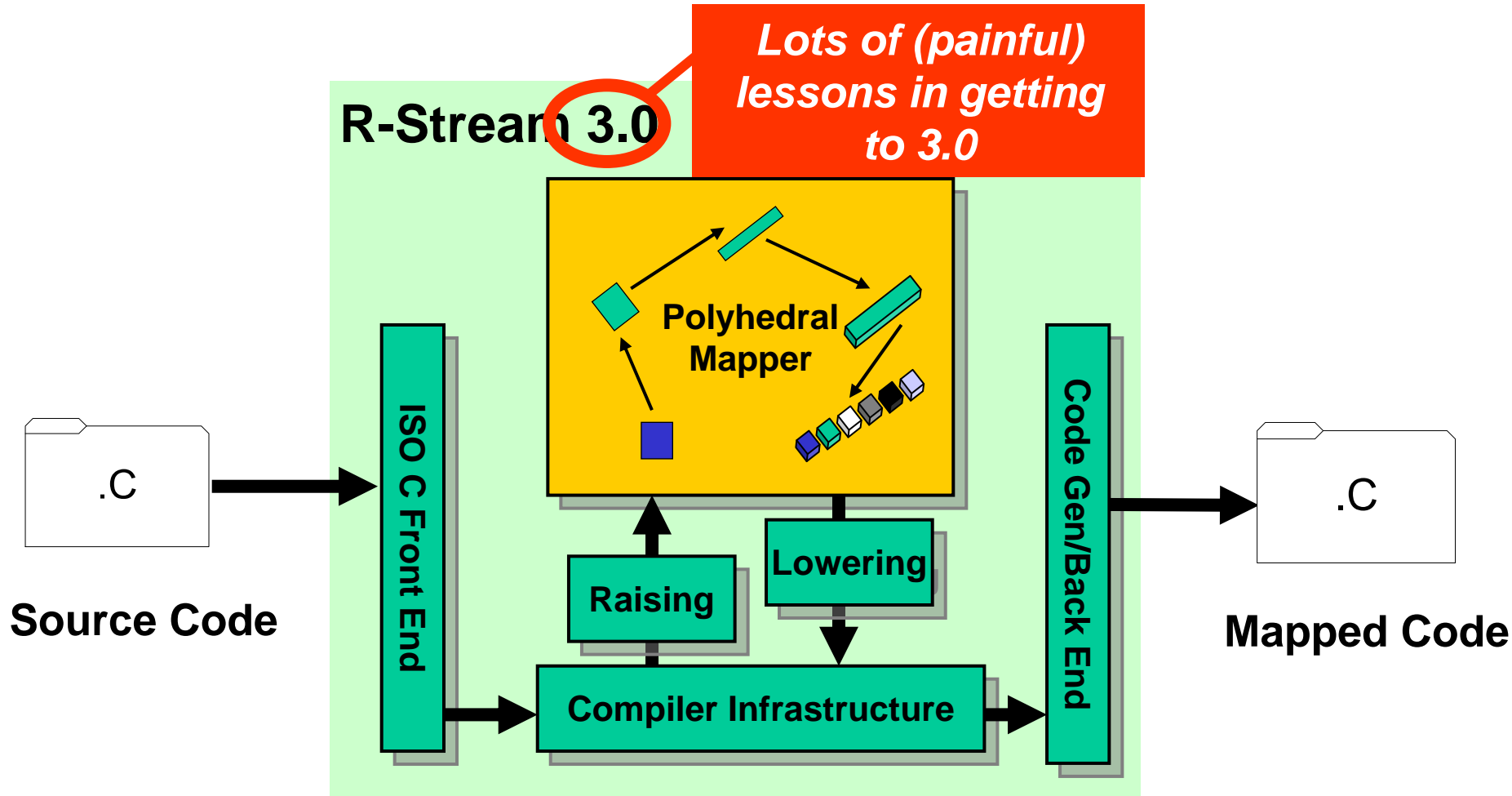
Register allocation

SIMDization

Instruction scheduling

Other scalar opts

Under the Hood of a High-Level Compiler



Polyhedral Representation

```

for (i=2; i<=M; i++) {
  for (j=0; j<=N; j+=2)
    A[i,N-j] = C[i-2,4*i+j/2];
  for (j=i; j<=N; j++)
    B[i,N-j] = A[i,j+1];
}

```

$$\{(i, j) \mid \exists k. 2 \leq i \leq M, 0 \leq j \leq N, j = 2k\}$$

Iteration spaces as constraints (polytopes)

$$\{(i, j) \mid 2 \leq i \leq M, i \leq j \leq N\}$$

**More precisely, Parametric Z-Domain:
(Intersection of lattice with parametric polytope)**

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ M \\ N \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} i & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} j \\ M \\ N \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} j \\ M \\ N \\ 1 \end{bmatrix}$$

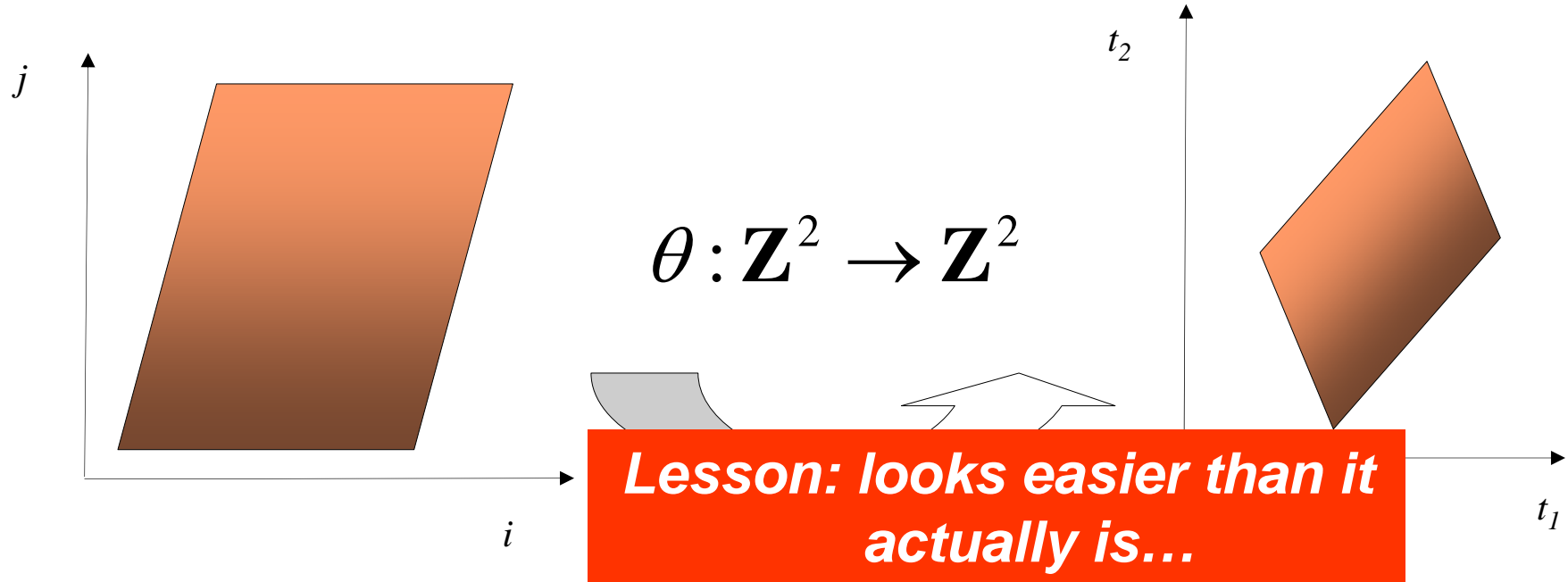
Array indices as affine functions

from these information.
Dependences are represented as polyhedra

Lesson: Must have a powerful mathematical basis for optimizations

Parallelization in the Polyhedral Model

iteration space of a statement $S(i,j)$



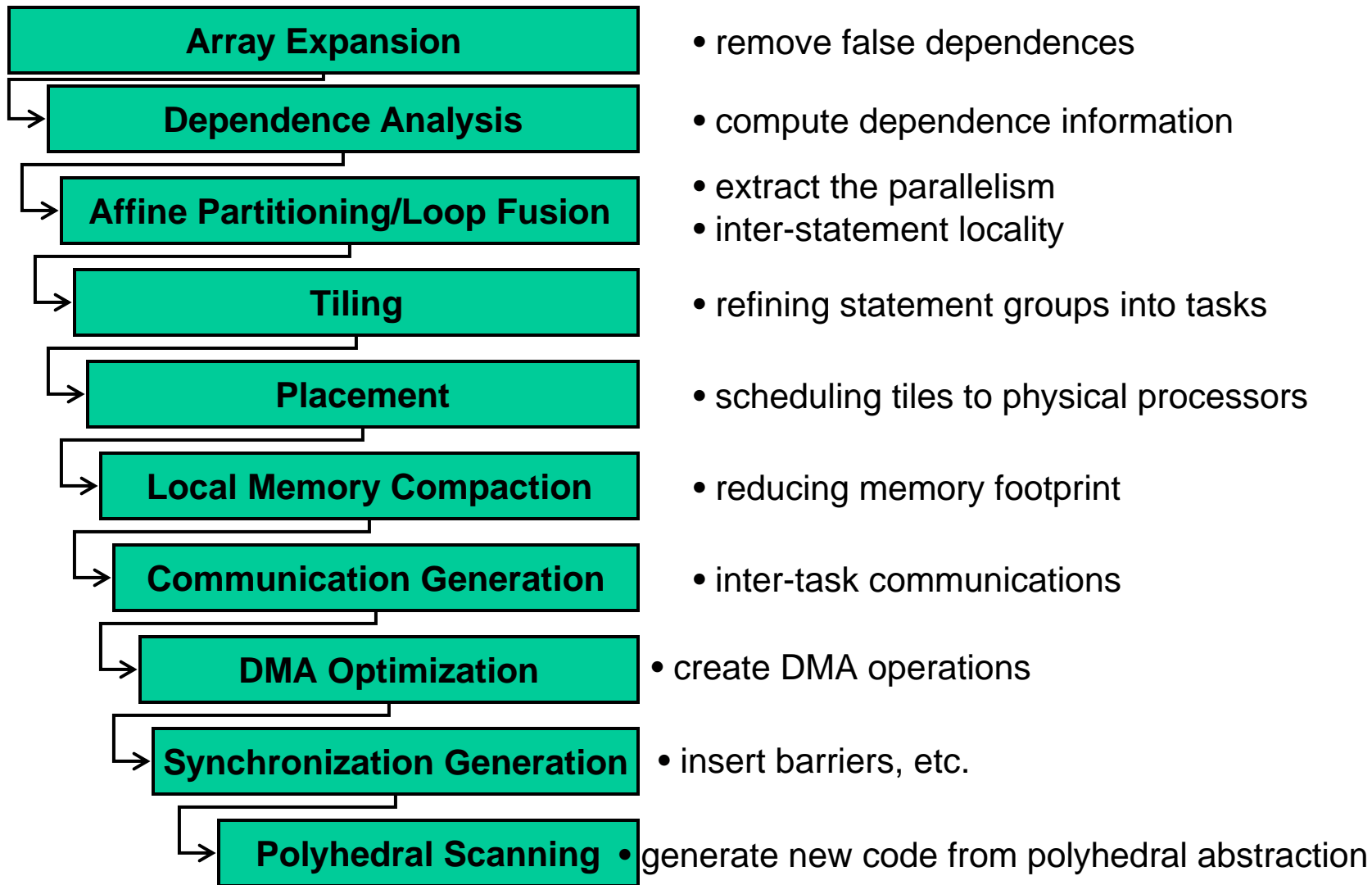
Schedule θ maps iterations to multi-dimensional time

Lesson: it can be done

Scheduling is choosing θ

Lesson: scalability

R-Stream Mapper Phase Ordering





Matrix Multiply Example

```
float A[1024][1024];
float B[1024][1024];
float C[1024][1024];
for (int i = 0; i <= 1023; i++) {
    for (int j = 0; j <= 1023; j++) {
        C[i][j] = 0;
        for (int k = 0; k <= 1023; k++) {
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
        }
    }
}
```

SPU Tiled Code for Matrix Multiply

```
for (k = -1; k <= 16; k++) { // 16 stages + 1 prologue and 1 epilogue
  if (k <= 15 && k >= 0) { // Block until the prefetched data is ready
    CELL_dma_wait(0);
    rotate C_l_v1 and C_l_v2, A_l_v1 and A_l_v2, B_l_v1 and B_l_v2;
  }
  if (k <= 14) {
    for (l = 0; l <= 63; l++) // Prefetch next block of A, B and C
      CELL_dma_get(&B[64*j+l][64+64*k], &B_l_v2[l][0], 64*4,4,4,1,0);
    for (l = 0; l <= 63; l++)
      CELL_dma_get(&A[512*i+l+64*PROC0][64*j], &A_l_v2[l][0], 64*4,4,4,1,0);
    for (l = 0; l <= 63; l++)
      CELL_dma_get(&C[512*i+l+64*PROC0][64+64*k], &C_l_v2[l][0], 64*4,4,4,1,0);
  }
  if (k <= 15 && k >= 0) { // 64x64 matrix multiply kernel
    doall (l = 0; l <= 63; l++)
      doall (m = 0; m <= 63; m++)
        for (n = 0; n <= 63; n++)
          C_l_v1[l][m] = C_l_v1[l][m] + B_l_v1[n][m] * A_l_v1[l][n];
  }
  if (k >= 1) CELL_dma_wait(1); // Block until the previous write completes
  if (k <= 15 && k >= 0) { // Initiate write back to C
    for (l = 0; l <= 63; l++)
      CELL_dma_put(&C_l_v1[l][0], &C[512*i+l+64*PROC0][64*k], 64*4,4,4,1,1);
  }
}
```

A Compilation Flow

```
for (int i = 0; i <= 255; i++) {  
  nrm[i] = 0;  
  for (int j = i; j <= 255; j++) {  
    nrm[i] = hypot(nrm[i], QR[j][i]);  
  }  
  if (nrm[i] != 0) {  
    nrm[i] = sn(QR[i][i], nrm[i]);  
    for (int j = i; j <= 255; j++) {  
      QR[j][i] = QR[j][i] / nrm[i];  
    }  
    QR[i][i] = 1 + QR[i][i];  
    for (int j = 1 + i; j <= 255; j++) {  
      s[i][j] = 0;  
      for (int k = i; k <= 255; k++) {  
        s[i][j] = s[i][j] + QR[k][i] * QR[k][j];  
      }  
      s[i][j] = - s[i][j] / QR[i][i];  
      for (int k = i; k <= 255; k++) {  
        QR[k][j] = QR[k][j] + QR[k][i] *  
        s[i][j];  
      }  
    }  
  }  
  Rdiag[i] = - nrm[i];  
}
```

ANSI C

**High-Level
Compiler**

API

**Low-Level
Compiler**

Cell

*Lesson: need a common
abstraction layer for Manycore
architecture*

Execution model

Communication

Computation

“Data Holes Problem” Influences Manycore API

```
for (i=0; i<=N; i++) {  
  for (j=0; j<=M; j++) {  
    A[i+4j] = ...  
  }  
}
```

The iteration domain is:

$$(i, j) \in Z^2 \mid 0 \leq i \leq N; 0 \leq j \leq M$$

It would seem like the “footprint” of A that is written by this loop is the image of the polyhedron bounding the iteration domain, i.e.,

$$0 \leq i \leq N; 0 \leq j \leq M$$

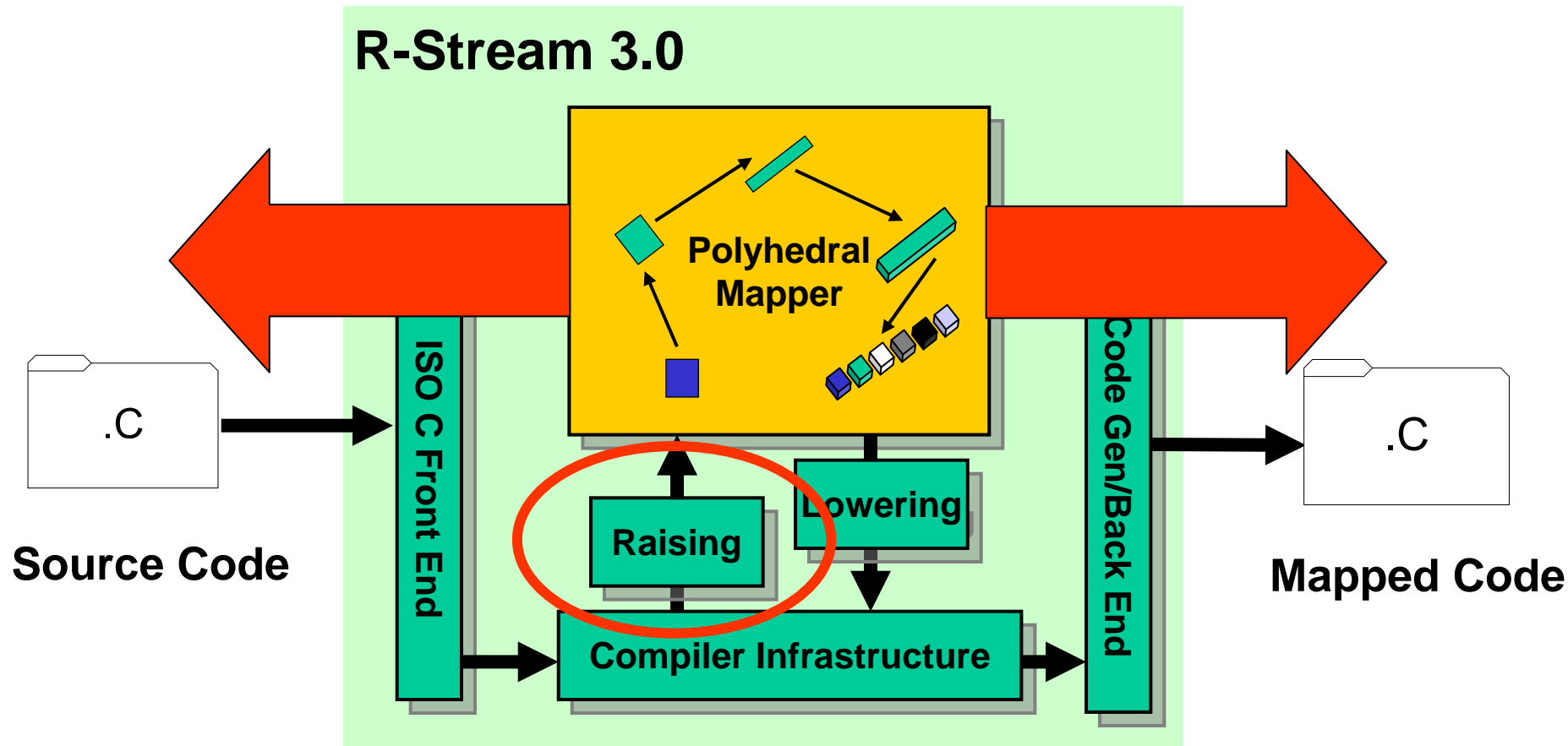
The API and maybe the hardware should reflect this mapper limitation

through $(i+4j)$, but because we are dealing with **parametric Z-domains**, it is something more complex when $N \leq 2$

Computing this statically is computationally very complex

The Compiler Determines Architecture

The Compiler Determines Language



The Future of Manycore

